DT505G: Algorithms, Data Structures and Complexity

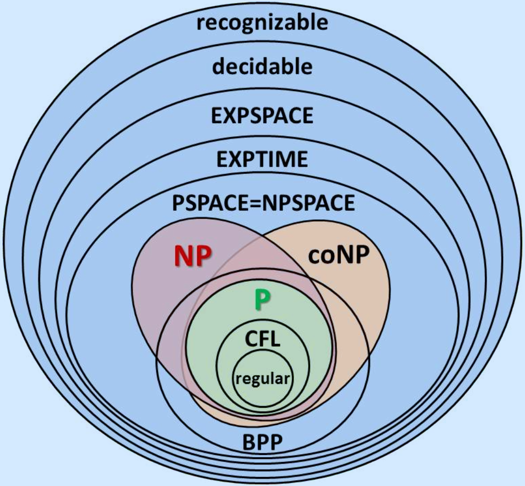
Introduction to Problem Complexity

Federico Pecora

federico.pecora@oru.se

Center for Applied Autonomous Sensor Systems (AASS)

Örebro University, Sweden



Tractable Problems

- Most of the algorithms we have seen run in $O(n^k)$ **time**
  - n is the size of the input
  - k is a constant
- Each algorithm solves a problem in polynomial time
- Are **all problems** solvable in polynomial time?

Tractable Problems

- Most of the algorithms we have seen run in $O(n^k)$ **time**
  - n is the size of the input
  - k is a constant
- Each algorithm solves a problem in polynomial time
- Are **all problems** solvable in polynomial time?
- **No**, some require more time
- Problems that can be solved in polynomial time are called **tractable**

Tractable Problems

- Most of the algorithms we have seen run in $O(n^k)$ **time**
  - n is the size of the input
  - k is a constant
- Each algorithm solves a problem in polynomial time
- Are **all problems** solvable in polynomial time?
- **No**, some require more time
- Problems that can be solved in polynomial time are called **tractable**
- Are problems that can be solved in $O(f(n))$ somehow **equivalent**?

Polynomial Time vs. Polynomial Space

- All the algorithms we have seen require $O(n^k)$ **space**
  - n is the size of the input
  - k is a constant
- The data structure maintained by the algorithm does not occupy more than a polynomial amount of space in the size of the input
- Are **all problems** solvable in polynomial space?

Polynomial Time vs. Polynomial Space

- All the algorithms we have seen require $O(n^k)$ **space**
  - n is the size of the input
  - k is a constant
- The data structure maintained by the algorithm does not occupy more than a polynomial amount of space in the size of the input
- Are **all problems** solvable in polynomial space?
- **No**, some require more space

Polynomial Time vs. Polynomial Space

- All the algorithms we have seen require $O(n^k)$ **space**
  - n is the size of the input
  - k is a constant
- The data structure maintained by the algorithm does not occupy more than a polynomial amount of space in the size of the input
- Are **all problems** solvable in polynomial space?
- **No**, some require more space
- Are problems that require $O(n^k)$ space **harder** than problems requiring $O(n^k)$ time?

Intractable Problems

- (Let's leave space complexity aside for now)
- Given a problem π, can we **prove that it cannot be solved** in time $O(n^k)$ for any constant k?

Intractable Problems

- (Let's leave space complexity aside for now)
- Given a problem π, can we **prove that it cannot be solved** in time $O(n^k)$ for any constant k?
- There exist many problems for which we **cannot** prove this
  - we know of algorithms that require more than polynomial time to solve π
  - we do not know whether a polynomial-time algorithm exists for solving π
- One set of such problems is the class of NP-**Complete** problems

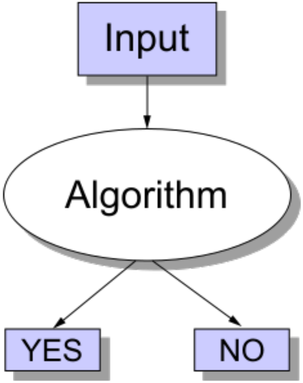
Intractable Problems: Example

- Shortest vs. longest simple paths in a graph $G = (V, E)$ from source $s \in V$
  - can compute a **shortest simple path** $s \rightsquigarrow v$ for all $v \in V$ in $O(VE)$ time
  - finding a **longest simple path** between two vertices is difficult
  - determining **whether G contains a simple path of at least k edges** is NP-Complete

Intractable Problems: Example

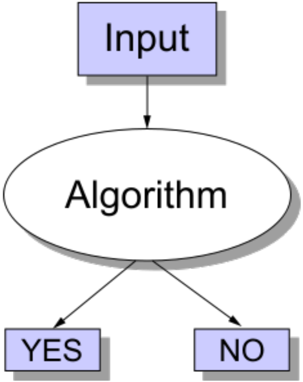
- Shortest vs. longest simple paths in a graph $G = (V, E)$ from source $s \in V$
  - can compute a **shortest simple path** $s \rightsquigarrow v$ for all $v \in V$ in $O(VE)$ time
  - finding a **longest simple path** between two vertices is difficult
  - determining **whether G contains a simple path of at least k edges** is NP-Complete
- Hamiltonian cycle of G: a simple cycle containing all $v \in V$
  - determining **whether G has a Hamiltonian cycle** is NP-Complete

Decision Problems



- A question in some formal system that can be posed as a yes-no question

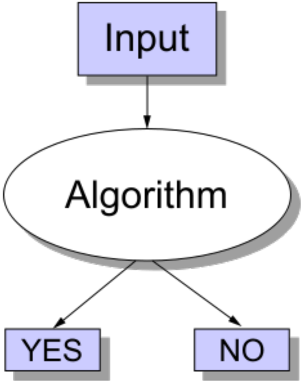
Decision Problems



- A question in some formal system that can be posed as a yes-no question

- **Problem:** "Find a shortest path in G from s to v"
- **Decision problem:** "Is there a path of at most length k in G from s to v?"

Decision Problems



- A question in some formal system that can be posed as a yes-no question

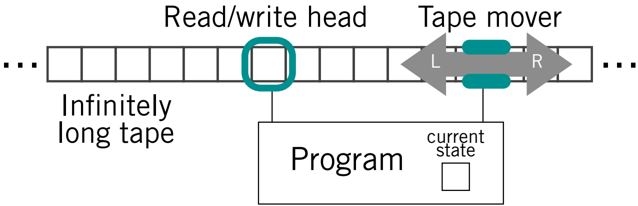
- **Problem:** “Find a shortest path in G from s to v”
- **Decision problem:** “Is there a path of at most length k in G from s to v?”

- **Problem:** “Find a negative cycle in G”
- **Decision problem:** “Is there a negative cycle in G?”

Alan Turing

- British computer scientist (1912–1954)
- Developed techniques for speeding the breaking of German ciphers in WWII
- Introduced a **computational model** that is used for studying the complexity of problems
- Considered a founder of Computer Science

Turing Machines



- A hypothetical machine whose program is a **transition function** δ
  - **states** Q**:** states
  - **states** $F \subseteq Q$**:** final/accepting states ($Q \setminus F$ = all non-final states)
  - **alphabet** Σ**:** symbols read/written by the head
  - **transitions:** $\delta : Q \setminus F \times \Sigma \mapsto Q \times \{\texttt{write(s)} \mid \texttt{s} \in \Sigma\} \times \{\texttt{moveL}, \texttt{moveR}\}$

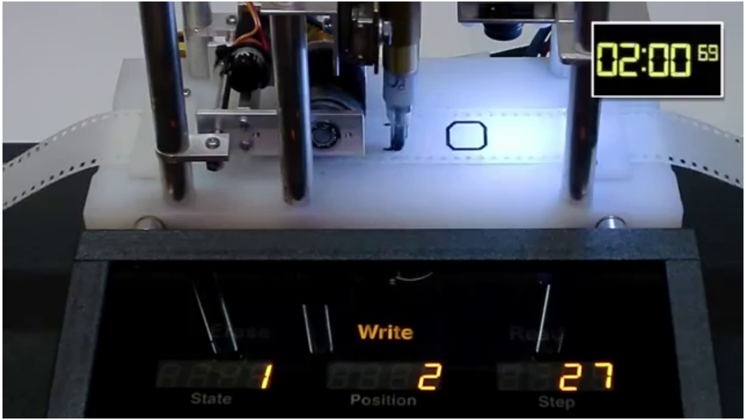
Turing Machines: Example

Transition function δ

State	Symbol read	Write instruction	Move instruction	Next state
A	`1`	`write(1)`	`moveR`	A
	`0`	`write(0)`	`moveR`	A
	`blank`	`write(blank)`	`moveL`	B
B	`0`	`write(1)`	`moveR`	A
	`1`	`write(0)`	`moveL`	B
	`blank`	`write(1)`	`moveR`	A

- $Q = \{\mathsf{A}, \mathsf{B}\}$, $F = \emptyset$, $\Sigma = \{0, 1\}$
- Implements a binary counter (continuously increments binary number on tape by one)

Turing Machines: Example



© 2010 Mike Davey — `http://aturingmachine.com`

Turing Machines: A Model of Computation

- A TM can comupte **anything a real computer can compute**
- A TM can manipulate **unbouded data** (the tape is infinite)
- TMs describe algorithms **independently of how much memory they use**

Turing Machines: A Model of Computation

- A TM can comupte **anything a real computer can compute**
  - limitation of TM is also a limitation of a computer
- A TM can manipulate **unbouded data** (the tape is infinite)

- TMs describe algorithms **independently of how much memory they use**

Turing Machines: A Model of Computation

- A TM can comupte **anything a real computer can compute**
  - limitation of TM is also a limitation of a computer
- A TM can manipulate **unbouded data** (the tape is infinite)
  - given a finite amount of time, a TM will use finite space
- TMs describe algorithms **independently of how much memory they use**

Turing Machines: A Model of Computation

- A TM can comupte **anything a real computer can compute**
  - limitation of TM is also a limitation of a computer
- A TM can manipulate **unbouded data** (the tape is infinite)
  - given a finite amount of time, a TM will use finite space
- TMs describe algorithms **independently of how much memory they use**
  - conclusions independent of advances in conventional computing machinery

Turing Machines: A Model of Computation

- A TM can solve **any problem** studied in this course
  $\rightsquigarrow$ *encode algorithm as transition function* δ
- Will a TM run an algorithm **less efficiently** than a RAM?

Turing Machines: A Model of Computation

- A TM can solve **any problem** studied in this course
  $\rightsquigarrow$ *encode algorithm as transition function* δ
- Will a TM run an algorithm **less efficiently** than a RAM?
- If a RAM requires time $\Theta(n)$, there is a TM that requires time $\Theta(n^6)$ [Rich, 2008]

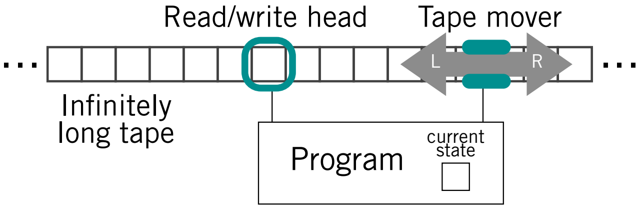
Turing Machines: A Model of Computation

- A TM can solve **any problem** studied in this course
  $\rightsquigarrow$ *encode algorithm as transition function* δ
- Will a TM run an algorithm **less efficiently** than a RAM?
- If a RAM requires time $\Theta(n)$, there is a TM that requires time $\Theta(n^6)$ [Rich, 2008]
- So, if our algorithm runs in polynomial time on a RAM, it can run in polynomial time on a TM

P = the set of all problems that can be solved by a TM in polynomial time

- All the problems we have studied are in P

Non-Deterministic Turing Machines



- A hypothetical machine whose program is a **relation** δ
  - **states** Q: states
  - **states** $F \subseteq Q$: final/accepting states ($Q \setminus F$ = all non-final states)
  - **alphabet** Σ: symbols read/written by the head
  - **relation:** $\delta \subseteq (Q \setminus F \times \Sigma) \times (Q \times \{\mathtt{write(s)} \mid \mathtt{s} \in \Sigma\} \times \{\mathtt{moveL}, \mathtt{moveR}\})$

Non-Deterministic Turing Machines: Example

Transition relation δ

State	Symbol read	Write instruction	Move instruction	Next state
A	1	`write(1)`	`moveR`	A
	...			
	0	`write(1)`	`moveL`	B
	0	`write(0)`	`moveR`	A
	...			
B	0	`write(1)`	`moveR`	C
	...			
...	...			

- $Q = \{A, B, \ldots\}$, $\Sigma = \{0, 1\}$
- Relation can prescribe **more than one (state,write,move) tuple for a given situation**

Non-Deterministic Turing Machines: Branching

- What does a NDTM do in a situation with > 1 action?

Non-Deterministic Turing Machines: Branching

- What does a NDTM do in a situation with > 1 action?
- NDTMs branch into **many copies**, each following one possible transition
- TMs have a **single computation path**
- NDTMs have **multiple computation paths**

Non-Deterministic Turing Machines: Branching

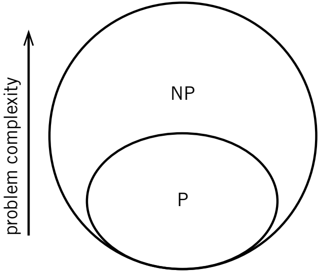
- What does a NDTM do in a situation with > 1 action?
- NDTMs branch into **many copies**, each following one possible transition
- TMs have a **single computation path**
- NDTMs have **multiple computation paths**
- Can NDTMs solve problems that TMs cannot?

Non-Deterministic Turing Machines: Branching

- What does a NDTM do in a situation with > 1 action?
- NDTMs branch into **many copies**, each following one possible transition
- TMs have a **single computation path**
- NDTMs have **multiple computation paths**
- Can NDTMs solve problems that TMs cannot?

No, NDTMs are only more efficient

Complexity Classes P and NP

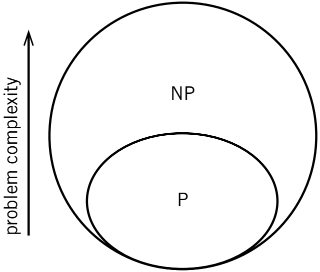


- $\pi \in$ P iff
  - π is a decision problem
  - π can be **solved** in **polynomial time by a TM**
- $\pi \in$ NP iff
  - π is a decision problem
  - π can be **solved** in **polynomial time by a NDTM**

Note: Solving vs. Verifying

- A NDTM can be seen as TM that "always guesses one course of action, but always guesses right"
- **Verifying** can be seen as "guessing post-facto"
- Proving that $\pi \in$ NP can be done by proving that you can **verify a solution to π in polynomial time**

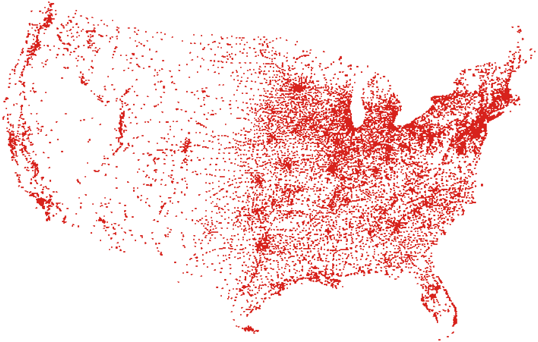
Complexity Classes P and NP



- $\pi \in$ P iff
  - π is a decision problem
  - π can be **solved** in **polynomial time by a TM**
- $\pi \in$ NP iff
  - π is a decision problem
  - $\mathrm{sol}(\pi)$ can be **verified** in **polynomial time by a TM**

The Traveling Salesperson Problem (TSP)

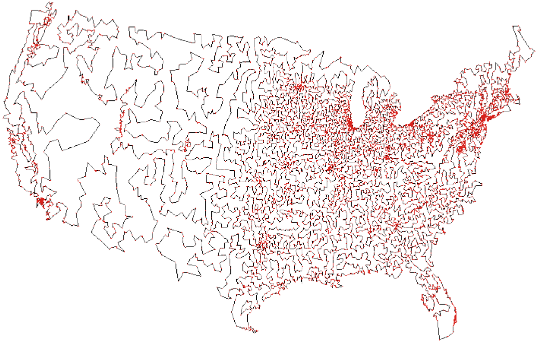
- Given a set of n cities C and a pairwise distance function $d : C^2 \mapsto \mathbb{R}$, is there a tour of length $\leq d_{\text{min}}$?



All 13509 cities in US with a population of at least 500 — `http://www.tsp.gatech.edu`

The Traveling Salesperson Problem (TSP)

- Given a set of n cities C and a pairwise distance function $d : C^2 \mapsto \mathbb{R}$, is there a tour of length $\leq d_{\text{min}}$?



Optimal tour — `http://www.tsp.gatech.edu`

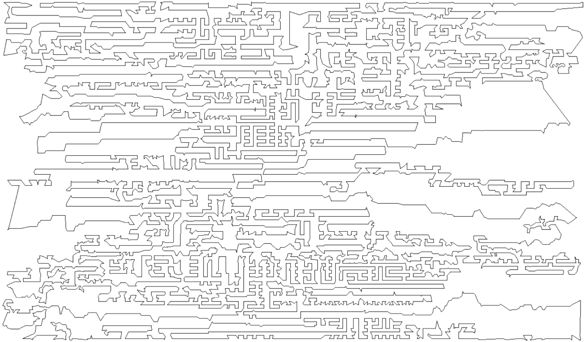
The Traveling Salesperson Problem (TSP)

- Given a set of n cities C and a pairwise distance function $d : C^2 \mapsto \mathbb{R}$, is there a tour of length $\leq d_{\text{min}}$?

11849 holes to drill in a programmed logic array — `http://www.tsp.gatech.edu`

The Traveling Salesperson Problem (TSP)

- Given a set of n cities C and a pairwise distance function $d : C^2 \mapsto \mathbb{R}$, is there a tour of length $\leq d_{\text{min}}$?



Optimal tour — `http://www.tsp.gatech.edu`

Proving that TSP $\in$ NP (Method 1)

- We can show a **non-determininstic algorithm** for solving TSP in **polynomial time**

TSP-DECIDE$(C, d : C^2 \mapsto \mathbb{R}, d_{\min})$

Create an empty sequence s
$W = C$
while $W \neq \emptyset$
    $c =$ CHOOSE(W)
    $W = W \setminus \{c\}$
    Add c to sequence s
// The sequence $s = \langle c_1, c_2, \ldots c_{|C|} \rangle$ now contains all cities
// Does it connect all of them in a loop with weight less than $d_{\min}$?
if $\sum_{i=1}^{|C|-1} d(c_i, c_{i+1}) + d(c_{|C|}, c_1) \leq d_{\min}$
    return TRUE
return FALSE

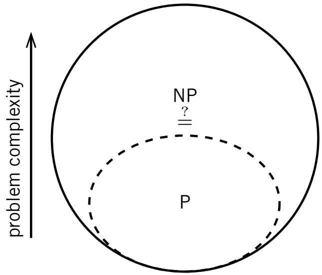
Proving that TSP $\in$ NP (Method 2)

- Assume we have a **certificate** $p = \langle C, d : C^2 \mapsto \mathbb{R}, d_{\min}, c_1, c_2, \ldots, c_n \rangle$ for a given TSP
- We can show a **determininstic algorithm** for verifying p in **polynomial time**

TSP-VERIFY(p)

if $n \neq |C|$ or $c_1 \neq c_n$ or $\sum_{i=1}^{n-1} d(v_i, v_{i+1}) + d(v_n, v_1) > d_{\min}$
    return FALSE
return TRUE

Complexity Classes P and NP

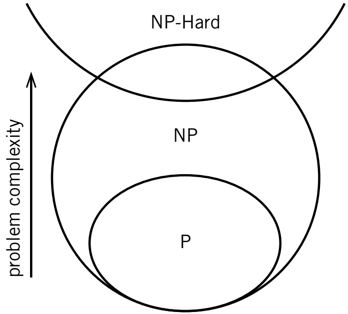


- $\pi \in$ P iff
  - π is a decision problem
  - π can be **solved** in **polynomial time by a TM**
- $\pi \in$ NP iff
  - π is a decision problem
  - $\mathrm{sol}(\pi)$ can be **verified** in **polynomial time by a TM**

P vs. NP

- Temporal complexity is **strongly curtailed** by non determinism
- However, we **cannot prove** that P $\neq$ NP
- We **strongly believe** that P $\neq$ NP
- The consequences of proving P $=$ NP would be huge
  - efficient algorithms would exist for all problems in NP
  - most cryptography systems would break
  - we could automatically prove any theorem which has a proof of reasonable length
  - many of the complexity classes would collapse into one

Complexity Classes P and NP



- $\pi \in$ P iff
  - π is a decision problem
  - π can be **solved** in **polynomial time by a TM**
- $\pi \in$ NP iff
  - π is a decision problem
  - sol(π) can be **verified** in **polynomial time by a TM**
- $\pi \in$ NP-Hard iff
  - all problems in NP can be **reduced** to π in **polynomial time**

Polynomial Time Reduction

- Problem π **polynomial-time reduces to** problem π' if arbitrary instances of π can be solved using
  - polynomial number of standard computational steps, plus
  - polynomial number of calls to an algorithm that solves problem π' in polynomial time
- In other words, we can solve π in polynomial time **if** we can solve π' in polynomial time

Polynomial Time Reduction

- Problem π **polynomial-time reduces to** problem π' if arbitrary instances of π can be solved using
  - polynomial number of standard computational steps, plus
  - polynomial number of calls to an algorithm that solves problem π' in polynomial time
- In other words, we can solve π in polynomial time **if** we can solve π' in polynomial time
- Hence, π' is **at least as hard** as π

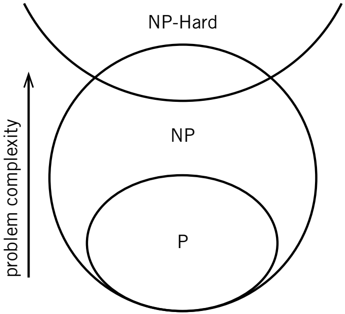
Polynomial Time Reduction: Example

- **Hamiltonian Cycle (HC):** Given a graph $G = (V, E)$, does there exists a simple cycle that contains every node in V?
- HC polynomial-time reduces to TSP:
  - Given $G = (V, E)$, create n cities with distance function

$$d(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 2 & \text{if } (u, v) \notin E \end{cases}$$

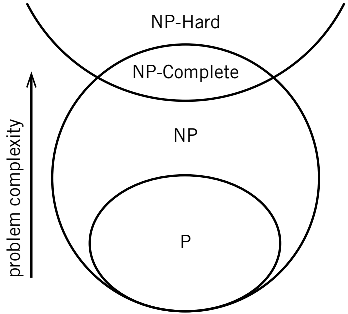
  - TSP has tour of length $\leq n$ iff G has a HC
- Hence, we can solve HC with an algorithm that solves TSP
- Hence, TSP is **at least as hard** as HC

Complexity Classes P and NP



- $\pi \in$ P iff
  - π is a decision problem
  - π can be **solved** in **polynomial time by a TM**
- $\pi \in$ NP iff
  - π is a decision problem
  - sol(π) can be **verified** in **polynomial time by a TM**
- $\pi \in$ NP-Hard iff
  - all problems in NP can be **reduced** to π in **polynomial time**

Complexity Classes P and NP

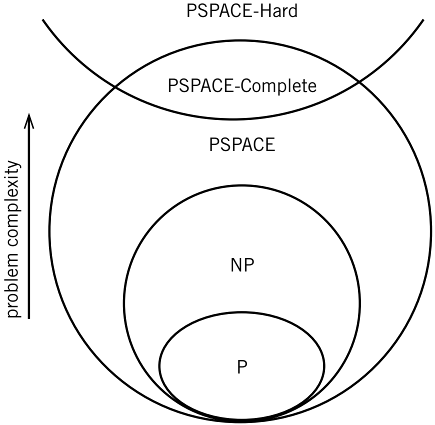


- $\pi \in$ P iff
  - π is a decision problem
  - π can be **solved** in **polynomial time by a TM**
- $\pi \in$ NP iff
  - π is a decision problem
  - sol(π) can be **verified** in **polynomial time by a TM**
- $\pi \in$ NP-Hard iff
  - all problems in NP can be **reduced** to π in **polynomial time**
- $\pi \in$ NP-Complete iff
  - $\pi \in$ NP and $\pi \in$ NP-Hard

Polynomial Time vs. Polynomial Space

- All the algorithms we have seen require $O(n^k)$ **space**
  - n is the size of the input
  - k is a constant
- The data structure maintained by the algorithm does not occupy more than a polynomial amount of space in the size of the input
- Are **all problems** solvable in polynomial space?
- **No**, some require more space

Complexity Class PSPACE



- $\pi \in$ PSPACE iff
  - π is a decision problem
  - π can be **solved** in **polynomial space by a TM**
- $\pi \in$ PSPACE-Hard iff
  - all problems in PSPACE can be **reduced** to π in **polynomial time**
- $\pi \in$ PSPACE-Complete iff
  - $\pi \in$ PSPACE and $\pi \in$ PSPACE-Hard

Spatial vs. Temporal Space Gians with NDTMs

- Spatial complexity is **not** strongly curtailed by non determinism
- It can be shown that PSPACE $=$ NPSPACE
  - if a NDTM can solve π using $f(n)$ space, a TM can solve π using $(f(n))^2$ space
  - if $f(n)$ is a polynomial, so is $(f(n))^2$
  - see [Savitch, 1970]
- However, the same **cannot be shown** for P and NP, that is, the question P $\stackrel{?}{=}$ NP remains open to this day

NP vs. PSPACE

- We can also show that NP $\subseteq$ PSPACE
  - a NDTM cannot use more than polynomial space in polynomial time, hence NP $\subseteq$ NPSPACE
  - since NPSPACE $=$ PSPACE, then NP $\subseteq$ PSPACE
  - see [Rich, 2008]

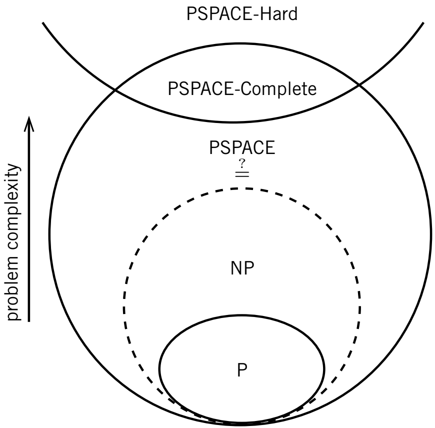
NP vs. PSPACE

- We can also show that NP $\subseteq$ PSPACE
  - a NDTM cannot use more than polynomial space in polynomial time, hence NP $\subseteq$ NPSPACE
  - since NPSPACE $=$ PSPACE, then NP $\subseteq$ PSPACE
  - see [Rich, 2008]
- Are problems that require $O(n^k)$ space **harder** than problems requiring $O(n^k)$ time?
- In other words, NP $\overset{?}{\subset}$ PSPACE

NP vs. PSPACE

- We can also show that NP $\subseteq$ PSPACE
  - a NDTM cannot use more than polynomial space in polynomial time, hence NP $\subseteq$ NPSPACE
  - since NPSPACE $=$ PSPACE, then NP $\subseteq$ PSPACE
  - see [Rich, 2008]
- Are problems that require $O(n^k)$ space **harder** than problems requiring $O(n^k)$ time?
- In other words, NP $\overset{?}{\subset}$ PSPACE
- We suspect that this is **true**, but nobody has proved it
- Problems that require more space **seem harder** than problems that require more time

Complexity Class PSPACE

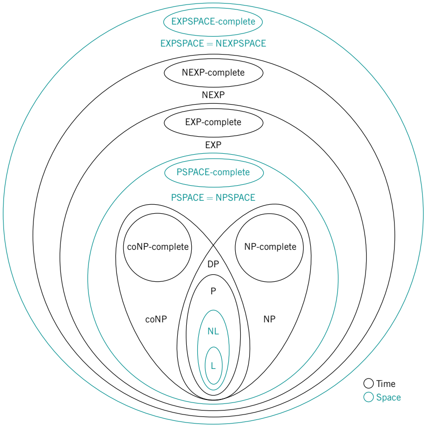


- $\pi \in$ PSPACE iff
  - π is a decision problem
  - π can be **solved** in **polynomial space by a TM**
- $\pi \in$ PSPACE-Hard iff
  - all problems in PSPACE can be **reduced** to π in **polynomial time**
- $\pi \in$ PSPACE-Complete iff
  - $\pi \in$ PSPACE and $\pi \in$ PSPACE-Hard

A Glimpse of the Wider Complexity Landscape

Class	*Computational model M*	*Time/Space requirement*	*Example(s)*
L	TM	$\mathrm{spacereq}(M) \in O(\log n)$	Majority
NL	NDTM	$\mathrm{spacereq}(M) \in O(\log n)$	Path existence
P	TM	$\mathrm{timereq}(M) \in O(n^k)$	Sorting, SSSP, APSP
NP	NDTM	$\mathrm{timereq}(M) \in O(n^k)$	TSP, Sudoku
PSPACE	TM	$\mathrm{spacereq}(M) \in O(n^k)$	Mahjong, Reversi
EXP	TM	$\mathrm{timereq}(M) \in O(2^{(n^k)})$	Chess, Checkers, Go
NEXP	NDTM	$\mathrm{timereq}(M) \in O(2^{(n^k)})$	...
EXPSPACE	TM	$\mathrm{spacereq}(M) \in O(2^{(n^k)})$	...

A Glimpse of the Wider Complexity Landscape



- PSPACE $=$ NPSPACE
- EXPSPACE $=$ NEXPSPACE
- NL $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE
- PSPACE $\subseteq$ EXP $\subseteq$ EXPSPACE
- NL $\subset$ PSPACE $\subset$ EXPSPACE
- P $\subset$ EXP

Thank you!

?

References

Rich, E. (2008).
Automata, computability and complexity: theory and applications.
Pearson Prentice Hall Upper Saddle River.

Savitch, W. J. (1970).
Relationships between nondeterministic and deterministic tape complexities.
Journal of computer and system sciences, 4(2):177–192.